It’s hard enough to find an error in your code when you’re looking for it; it’s even harder when you’ve assumed your code is error-free.
~ *Steve McConnell*

CS1010 Programming Methodology

Week 8: Two-dimensional Arrays

I. Basics

1. Given the following program **q1.c**

```
#include <stdio.h>
#define MAX_ROW 2
#define MAX_COL 3

void printArray(int [][], int, int);

int main(void) {
   int values[MAX_ROW][MAX_COL];
   int row, col;

   printf("Enter values: \n");
   for (row=0; row<MAX_ROW; row++)
      for (col=0; col<MAX_COL; col++)
         scanf("%d", values[row][col]);

   printf("Array entered contains:\n");
   printArray(values, MAX_ROW, MAX_COL);

   return 0;
}

void printArray(int arr[][], int row_size, int col_size) {
   int row, col;

   for (row=0; row<row_size; row++) {
      for (col=0; col<col_size; col++)
         printf("%d ", arr[row][col]);
      printf("\n");
   }
}
```

(a) Spot the errors in the program.

(b) After you have corrected the errors in the program, run it by entering the data in each of the following formats. Do they work? What can you deduce?

```
8 1 2
3 0 9
```

```
  8   1     2
3   0         9
```

```
8 1
2 3
0 9
```

```
8
1 2 3

0 9
```

```
8
1
2
3
0
9
```

2. Modify your corrected program for Q1 above by changing the int array into a **char array**, and changing the **%d** format specifier in the **printf()** and **scanf()** statements to **%c**.

   Run your program on the following input. Does it work? If not, why and how would you correct the program in a simple way without having to rewrite much of the code?

   ```
   abc
   def
   ```

3. **Manual tracing**: Write the output for each of the code fragments below.

   (a) This is adapted from CS1101 exam paper AY2006/7 Semester 1.
   Trace it out manually before you verify your answer by running **q3a.c**.

   ```
   int sum[4][4], k, m, n;

   for (k=0; k<4; k++) sum[k][0] = 1;
   for (k=0; k<4; k++) sum[0][k] = 1;

   for (m=1; m<4; m++)
      for (n=1; n<4; n++)
         sum[m][n] = sum[m-1][n] + sum[m][n-1];

   for (n=1; n<4; n++)
      printf("%d ", sum[3][n]);
   printf("\n");
   ```

(b) This is adapted from CS1101 exam paper AY2008/9 Semester 1.
Trace it out manually before you verify your answer by running **q3b.c**.

```
int array[][3] = { {1,1,1}, {2,2,2}, {3,3,3} };
int i, j;

for (i=0; i<3; i++)
   for (j=0; j<3; j++)
      array[i][j] += array[j][i];

for (i=0; i<3; i++) {
   for (j=0; j<3; j++)
      printf("%d ", array[i][j]);
   printf("\n");
}
```

II. Programming Exercises

To students:

The following questions are from Practice Exercises. Your DL may give you additional questions to attempt.

You may want to write many programs on 2D array for practice. The program **random_2Darray.c** provided might come in handy as it assigns random non-negative integers into a 2D array for you to work on later. You may adapt it.

4. To multiply two matrices A and B, the number of columns in A, let's call it n, must be the same as the number of rows in B. The resulting matrix has the same number of rows as A and number of columns as B. Hence, multiplying a $k \times n$ matrix with an $n \times p$ matrix gives a $k \times p$ product matrix.

   To compute C = A × B, where A, B, C are matrices,

   $$C_{i,j} = (A_{i,0} \times B_{0,j}) + (A_{i,1} \times B_{1,j}) + \ldots + (A_{i,n-1} \times B_{n-1,j})$$

   Two examples are shown here:

   $$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} -1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 2 & -1 \end{pmatrix} = \begin{pmatrix} 3 & 2 & 0 \\ 2 & 3 & -1 \\ -1 & 2 & -1 \end{pmatrix}$$

   $$\begin{pmatrix} 2 & 1 & 3 & 2 \\ 3 & 0 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 & 2 & 1 \\ 2 & 2 & 3 \\ 1 & 3 & 0 \\ 2 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 15 & 17 & 11 \\ 13 & 13 & 6 \end{pmatrix}$$

   Write a function to perform the product of two matrices. You may assume that a matrix has at most 10 rows and 10 columns.

5. A **square matrix** is a two-dimensional array where the number of rows and columns are the same. Write a program **square_matrix.c** to read in values for an $n \times n$ square matrix containing integer values, and check whether the matrix is (a) a diagonal matrix, or (b) an upper-triangular matrix.

   A **diagonal matrix** is a square matrix in which the elements outside the main diagonal (⭸) are all zeroes, for example:

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -2 \end{bmatrix} \qquad \begin{bmatrix} 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -5 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix}$$

   An **upper triangular matrix** (or right triangular matrix) is a square matrix U of the form:

$$U_{ij} = \begin{cases} a_{ij} & \text{for } i \le j \\ 0 & \text{for } i > j. \end{cases}$$

   Written explicitly,

$$U = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

   Note that a diagonal matrix is also an upper triangular matrix.

   A sample run is shown below. The first line contains a single integer indicating the size of the square matrix, n. The next $n \times n$ values are the elements of the matrix. The output is in bold. You may assume that the matrix contains at most 10 rows and 10 columns.

```
5
2 -1 3 4 1
0 7 5 -2 0
0 0 6 0 4
0 0 0 0 8
0 0 0 0 2
Matrix read:
   2  -1   3   4   1
   0   7   5  -2   0
   0   0   6   0   4
   0   0   0   0   8
   0   0   0   0   2
Matrix is not a diagonal matrix.
Matrix is an upper triangular matrix.
```

   You may download the incomplete program **square_matrix.c** from the module website (under “CA” → “Discussion”), or copy it over to your directory with this command:

   cp ~cs1010/discussion/prog/week7/square_matrix.c .

   Complete the program.

III. Worksheet

Read up Lab #4 assignment on the module website:

http://www.comp.nus.edu.sg/~cs1010/3_ca/labs.html

A worksheet on Lab #4 Exercise 2 will be given out during the discussion session.